Algorithm Analysis

CSC 209 Data Structures

Kulwadee Somboonviwat

kulwadee.som [at] sit.kmutt.ac.th

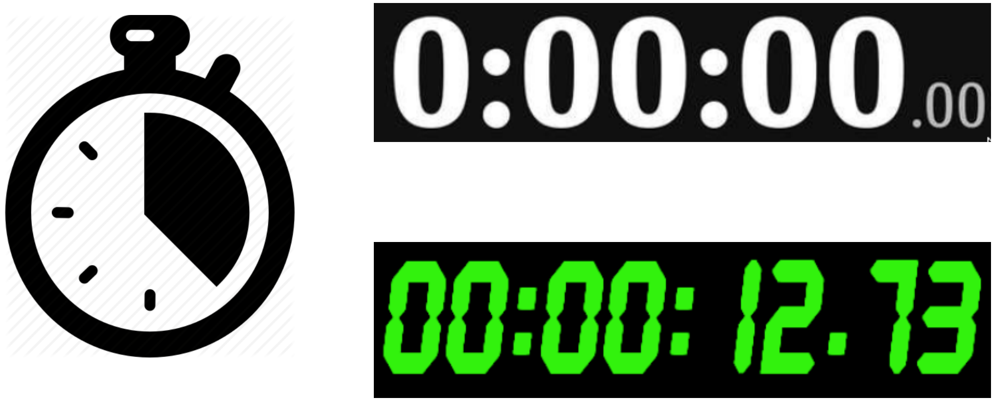
Motivating Example

- both ArrayList and LinkedList implement the List interface
- How to decide which one is better?
  - Profiling
    - Must implement two versions of the same program
    - Performance evaluation results might depend on the HARDWARE
    - Performance evaluation results might depend on the size and characteristics of the DATA
  - Analysis of algorithms
    - To avoid “implementation” and HARDWARE dependence

      Identify the basic operations that make up an algorithm (e.g. addition, multiplication, comparison) and count the number of each operations each algorithm requires.
    - To avoid DATA dependence, analyze the average case or worst case scenarios

Profiling Performance of ArrayList and LinkedList

co.kulwadee.csc209.lect03. JCFListPerformance




```java
public static long testListAdd(List<Integer> list) {
    int numIters = 100000;
    long startTime, endTime, duration;
    startTime = System.nanoTime();
    for (int i = 0; i < numIters; i++) {
        list.add(i);
    }
    endTime = System.nanoTime();
    duration = endTime - startTime;
    return duration;
}
 // ArrayList add
 duration = testListAdd(arrayList);
 strDuration = formatDecimal(nanoToSeconds(duration));
 System.out.println(" ArrayList add: " + strDuration + " s");

 // LinkedList add
 duration = testListAdd(linkedList);
 strDuration = formatDecimal(nanoToSeconds(duration));
 System.out.println("LinkedList add: " + strDuration + " s");
```

Profiling Performance of SelectionSort and InsertionSort

co.kulwadee.csc209.lect03. SelectionSort

```
public static long testSelectionSort(int[] array) {
    long startTime, endTime;
    startTime = System.nanoTime();
    selectionSort(array);
    endTime   = System.nanoTime();
    return endTime - startTime;
}
```

Array size: Elapsed Time

```
16:  0.00018
32:  0.00019
64:  0.00063
128:  0.00331
256:  0.00382
512:  0.01417
1024:  0.04982
```

co.kulwadee.csc209.lect03. InsertionSort

```
public static long testInsertionSort(int[] array) {
    long startTime, endTime;
    startTime = System.nanoTime();
    insertionSort(array);
    endTime   = System.nanoTime();
    return endTime - startTime;
}
```

Array size: Elapsed Time

```
16:  0.00008
32:  0.00016
64:  0.00065
128:  0.00209
256:  0.00742
512:  0.01794
1024:  0.02808
```

N	selection sort	insertion sort
16	0.00018	0.00008
32	0.00019	0.00016
64	0.00063	0.00065
128	0.00331	0.00209
256	0.00382	0.00742
512	0.01417	0.01794
1024	0.04982	0.02808

Analyzing Selection Sort

Identify the basic operations that make up an algorithm and count the number of each operations each algorithm requires.

```java
/**
 * Swaps the elements at index i and j.
 */
public static void swapElements(int[] array, int i, int j) {  // constant
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}

/**
 * Finds the index of the lowest value
 * starting from the index at start (inclusive)
 * and going to the end of the array.
 */
public static int indexLowest(int[] array, int start) {       // linear
    int lowIndex = start;
    for (int i = start; i < array.length; i++) {
        if (array[i] < array[lowIndex]) {
            lowIndex = i;
        }
    }
    return lowIndex;
}

/**
 * Sorts the elements (in place) using selection sort.
 */
public static void selectionSort(int[] array) {                // quadratic
    for (int i = 0; i < array.length; i++) {
        int j = indexLowest(array, i);
        swapElements(array, i, j);
    }
}
```

Big-Oh Notation

- Provides a convenient way to write general rules about how algorithms behave when we compose them.
- For example, if you perform a linear time algorithm followed by a constant algorithm, the total run time is linear.

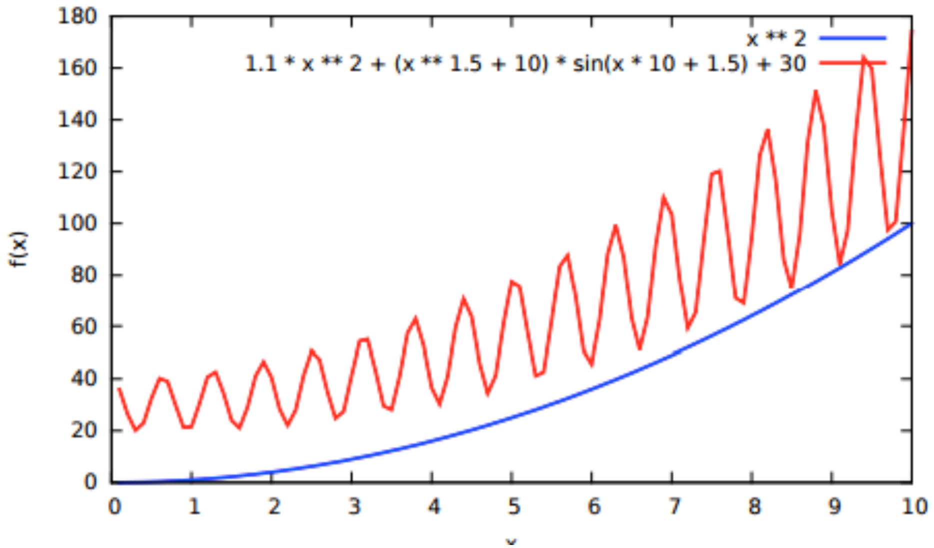
$$\text{If } f \in O(n) \text{ and } g \in O(1), f+g \in O(n).$$

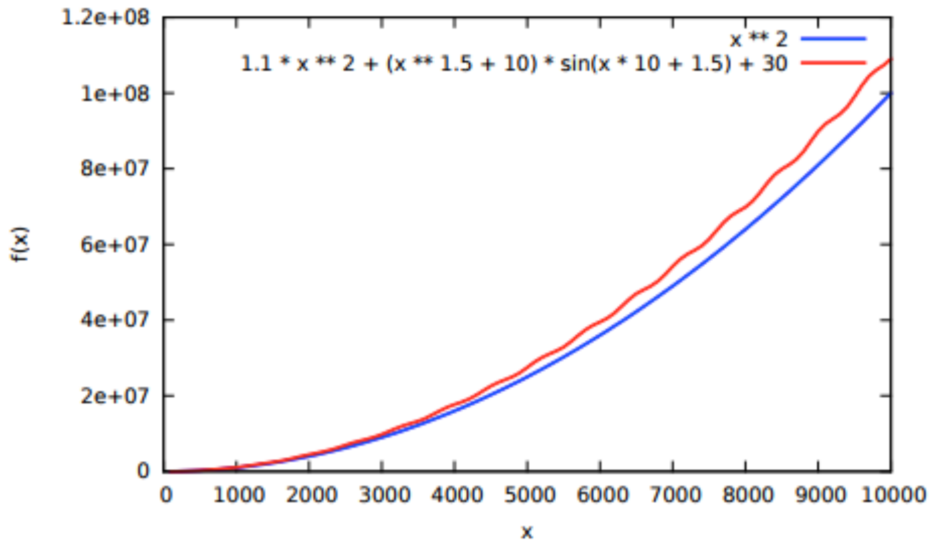
- If you perform two linear operations, the total is still linear:

$$\text{If } f \in O(n) \text{ and } g \in O(n), f+g \in O(n).$$

Asymptotic Complexity

$$f_1(x) = x^2 \qquad f_2(x) = 1.1x^2 + (x^{1.9} + 10)\sin(10x + 1.5) + 30$$

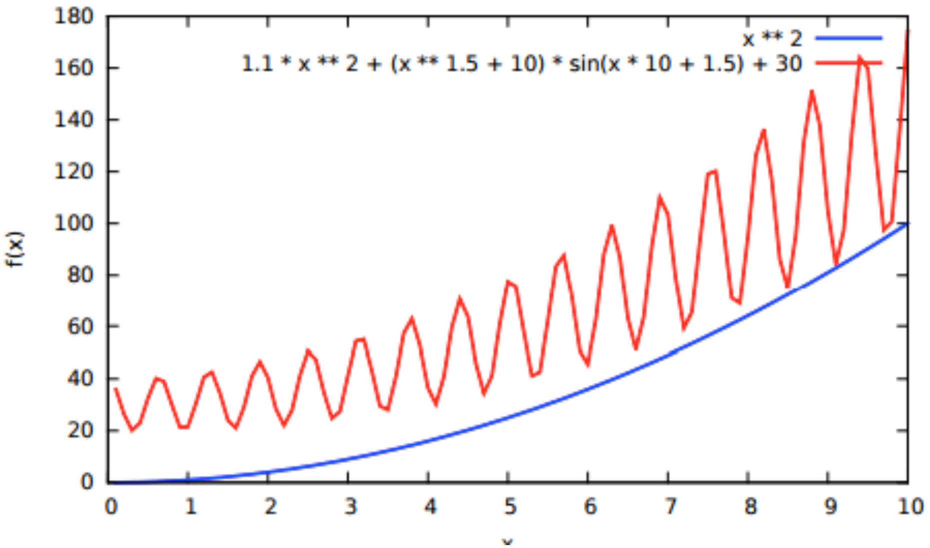


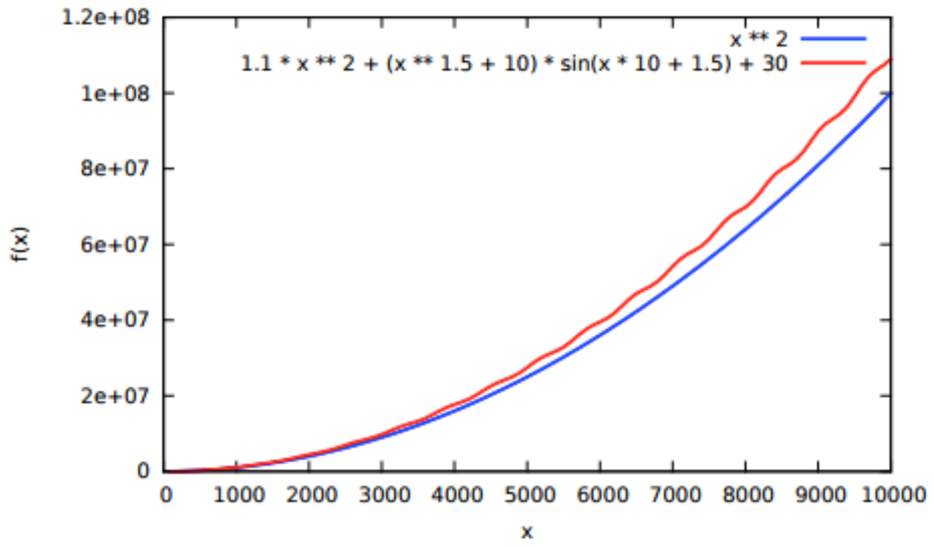


This example was taken from: https://goo.gl/AzXo6t

Asymptotic Complexity

$$f_1(x) = x^2 \qquad f_2(x) = 1.1x^2 + (x^{1.9} + 10)\sin(10x + 1.5) + 30$$



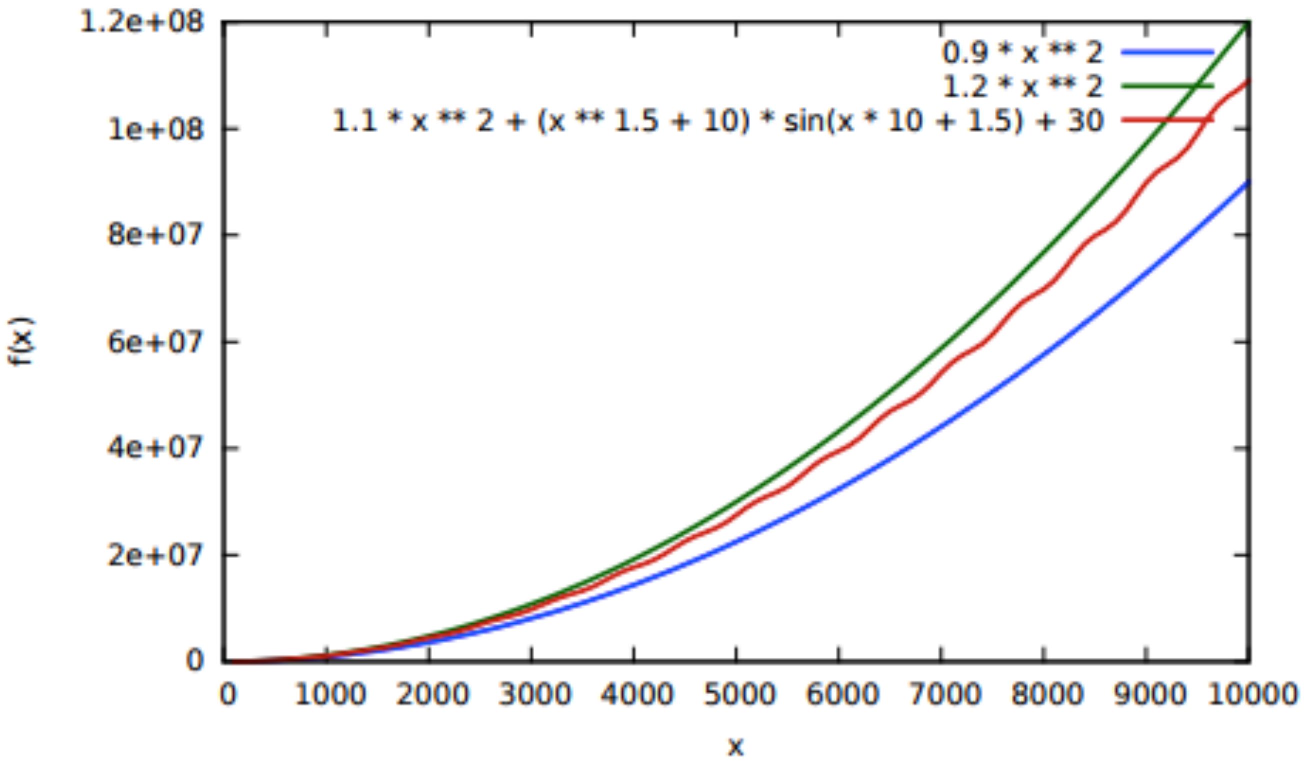


*This example was taken from: https://goo.gl/AzXo6t

Asymptotic Notation – Theta Θ

$$\boxed{f_2(x) = 1.1x^2 + (x^{1.9} + 10)\sin(10x + 1.5) + 30}$$

$$\boxed{f_2(x) = \Theta(x^2)}$$ => f_2 is constrained both from above and below by x^2



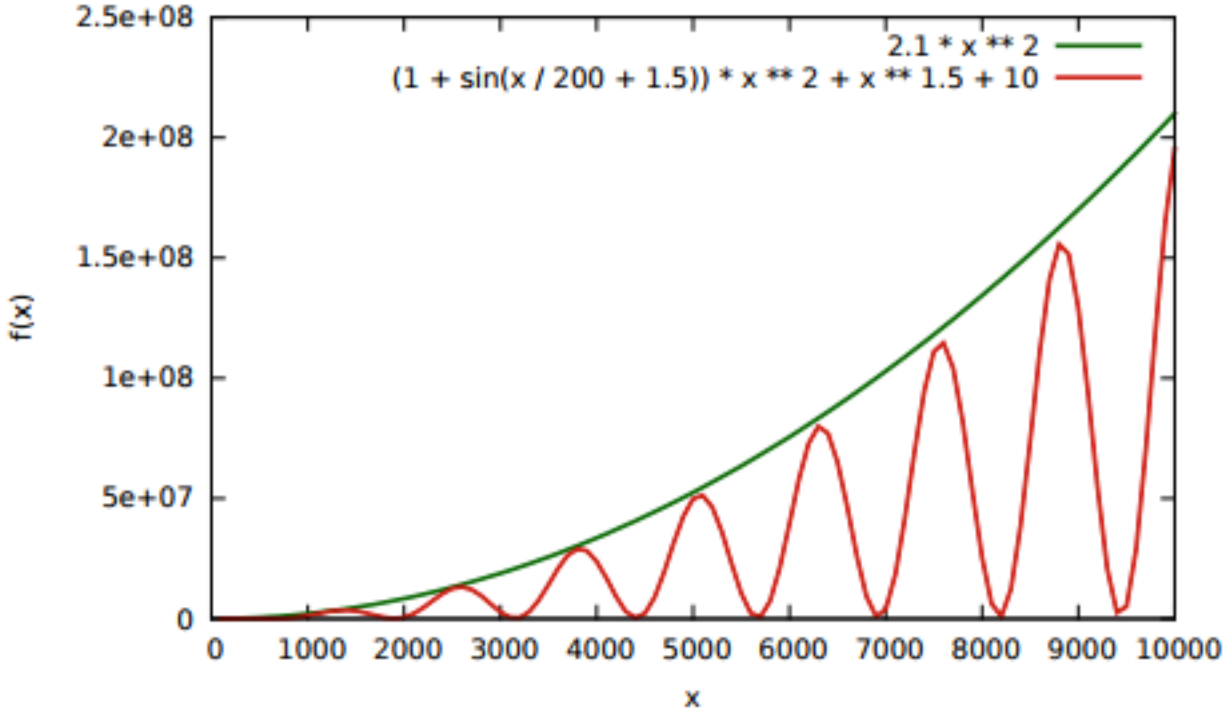
**This example was taken from: https://goo.gl/AzXo6t*

Asymptotic Notation – Big-Oh *O*

$$g(x) = (1 + \sin(\frac{x}{200} + 1.5))x^2 + x^{1.5} + 10$$

$$\boxed{g(x) = O(x^2)}$$

=> *g* is constrained from above by x^2



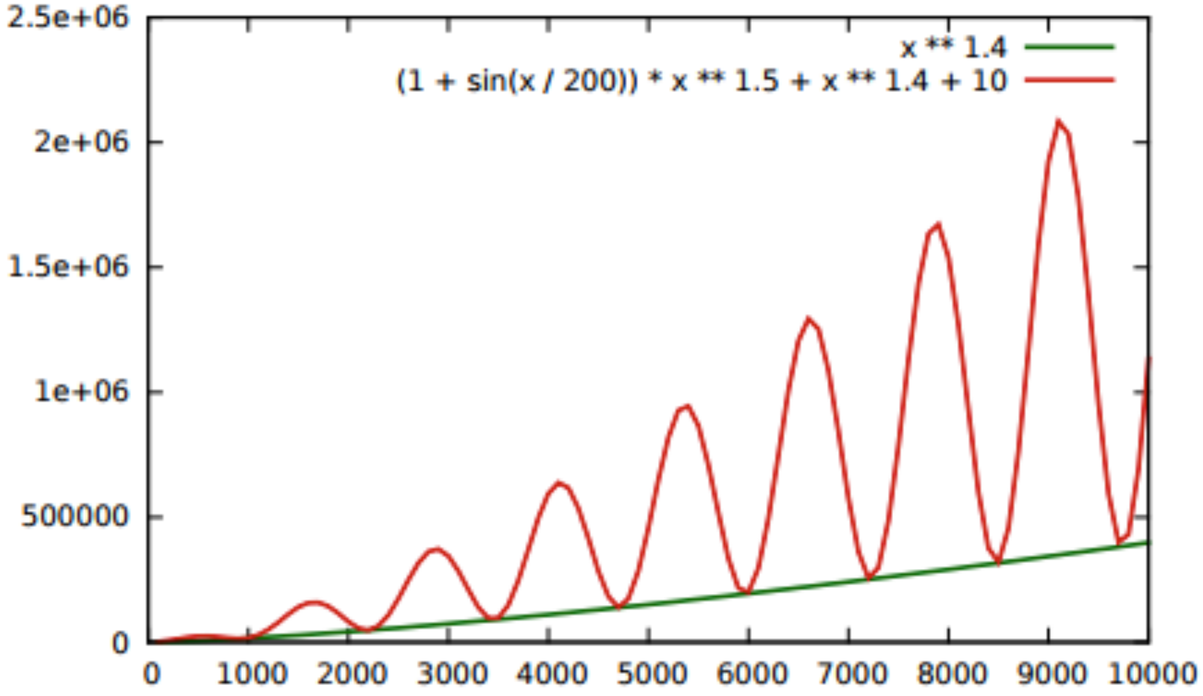
**This example was taken from: https://goo.gl/AzXo6t*

Asymptotic Notation – Omega Ω

$$h(x) = (1 + \sin(\tfrac{x}{200} + 1.5))x^{1.5} + x^{1.4} + 10,$$

$$\boxed{h(x) \;=\; \Omega(x^{1.4})}$$

=> *h* is constrained from below by x^2



This example was taken from: https://goo.gl/AzXo6t

Common Functions when Analyzing Algorithms

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$	[*see page 47*]	*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`   if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$	[*see* ALGORITHM 2.4]	*divide and conquer*	*mergesort*

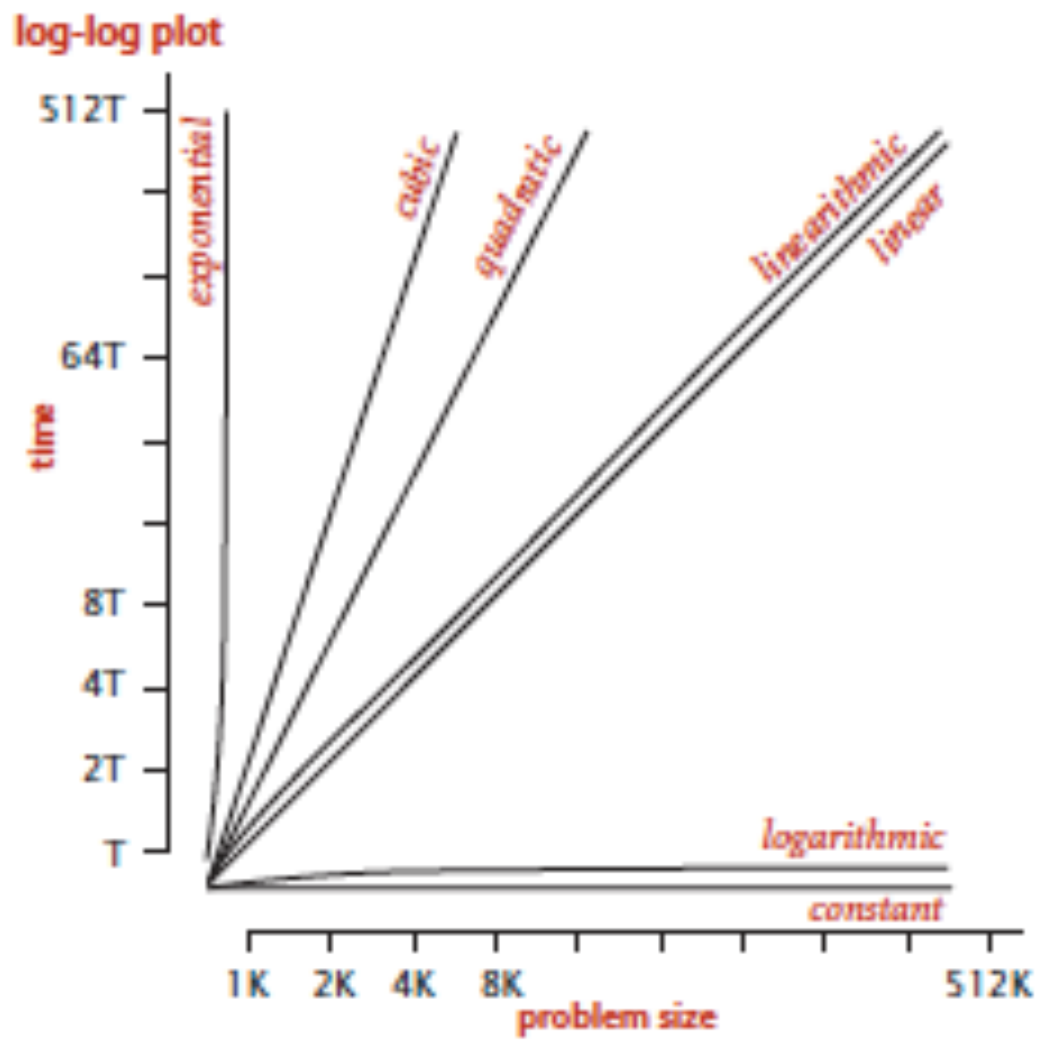
Common Functions when Analyzing Algorithms

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`    if (a[i] + a[j] == 0)`
		`      cnt++;`
			double loop	*check all pairs*

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`    for (int k = j+1; k < N; k++)`
		`      if (a[i] + a[j] + a[k] == 0)`
		`        cnt++;`
			triple loop	*check all triples*

exponential	2^N	[*see* CHAPTER 6]	*exhasutive search*	*check all subsets*

Common Functions when Analyzing Algorithms



Typical orders of growth

Pop-Quiz

Order the following functions by asymptotic growth rate

$$
\begin{array}{ccc}
4n\log n + 2n & 2^{10} & 2^{\log n} \\
3n + 100\log n & 4n & 2^n \\
n^2 + 10n & n^3 & n\log n
\end{array}
$$